Instrument 2.0

A.K.A Beamline

Why Do We Need Geometry?

Just a few applications...

- Unit conversion
- Absorption correction
- Solid angle correction
- Step scanning
- Visualisation

Mantid provides in-memory virtual geometry. This gives runtime-control!

C++ Side

This presentation will not cover the c++ side of the instrument 2.0 codebase. Link below is a good start.

https://github.com/mantidproject/mantid/tree/master/Framework/Beamline

Brief Timeline

1. **2007** Instrument 1.0
2. **2015** Performance report + ORNL 5-year plan[1]
3. **2016** Prototype(s)
4. **2017** COW Flat Tree Approach
5. **2018** Widespread Mantid Application

> *Using Instrument 2.0 gives 100x read performance and 10x read performance over Instrument 1.0!*

[1] https://github.com/mantidproject/documents/blob/master/Design/ORNL_Mantid_5yearplan.pdf

Common Principles In Classes

1. Optimise for Read access
2. Copies are first-class
3. Indexing for programmatic access
4. Iterator support
5. Virtualisation avoided

What Is It

Comprises three main classes...

SpectrumInfo

DetectorInfo

ComponentInfo

What Is It

Comprises three main classes...

SpectrumInfo

DetectorInfo

ComponentInfo

What Is It

Comprises three main classes...

SpectrumInfo

DetectorInfo

ComponentInfo

What Is It

Comprises three main classes...

SpectrumInfo

DetectorInfo

ComponentInfo

SpectrumInfo

Enumerable type supporting access to spectrum level properties, and accounting for detector groupings in internal operations. Comprises zero or more detectors.

SpectrumInfo

Access spectra level properties

```
ws = CreateSampleWorkspace()
spec_info = ws.spectrumInfo()
spec_info.l2(0)
```

SpectrumInfo Iteration

```
ws = CreateSampleWorkspace()
spec_info = ws.spectrumInfo()
for spec in spec_info:
    if spec.isMonitor:
        spec.setMasked
```

DetectorInfo

Enumerable type supporting access to detector level properties and geometric operators

DetectorInfo

```
detinfo = ws.detectorInfo()
detinfo.twoTheta(0)
```

DetectorInfo Iteration

```
ws = CreateSampleWorkspace()
det_info = ws.detectorInfo()
for det in det_info:
    if det.isMonitor:
        det.setMasked
```

ComponentInfo

Flat tree representation of the instrument

ComponentInfo - Operations

```
ws = CreateSampleWorkspace()
comp_info = ws.componentInfo()
bank_index = comp_info.indexOfAny('bank1')
print(bank_index)
# 210
print(len(comp_info.componentsInSubtree(bank_index)))
# 111
print(len(comp_info.detectorsInSubtree(bank_index)))
# 100
print(comp_info.parent(bank_index))
# 225
print(comp_info.parent(comp_info.root()))
# 225
```

ComponentInfo - Index Ranges

Essential points...

1. Every detector index is a valid component index
2. The component index for a detector is EQUAL to the detector index
3. Detector index ranges go from 0 - N
4. <u>Non Detector</u> component index ranges go from N - M
5. Root always has highest index

ComponentInfo - Index Ranges Example

```
ws = CreateSampleWorkspace()
comp_info = ws.componentInfo()
det_info = ws.detectorInfo()
print(comp_info.root())
# 225
print(len(comp_info))
# 256
print(len(det_info))
# 220
print(comp_info.isDetector(len(det_info)-1))
# True : detector range
print(comp_info.isDetector(len(det_info)-1))
# True : detector range
print(comp_info.isDetector(len(det_info)))
# False : - non-detector component ranges
```

Guideline: 1

Treat getInstrument() on a MatrixWorkspace as deprecated

```
ws.getInstrument() # Old API
```

Guideline: 2

One should NEVER expose a Component Index or Detector Index through a user facing interface, such an algorithm or fit function

Guideline: 3

Programatically use indexes, convert IDs to indexes early if necessary

Further Information

https://docs.mantidproject.org/nightly/concepts/InstrumentAccessLayers.htm